VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Simulační algoritmus SVD pro efektivní analýzu rozsáhlých dat

A simulation algorithm of SVD for effective analysis of large-scale data

2013 Lukáš Pištělák

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 6. dubna 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. dubna 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Tato práce se zabývá efektivním využitím simulační metody Monte-Carlo s aplikací na částečný SVD rozklad, který je metodou LSI-latentní sémantické indexovaní, dále používaný. SVD rozklad je robustní a efektivní matematický rozklad, je užitečný na všechny problémy týkající se např. při kompresi obrazových dat a k vyhledávání podobnosti mezi dokumenty, obrázky atd. Přímý výpočet SVD je velmi náročný na procesor a paměť, hlavně při rozsáhlejších datových maticích. Nicméně multimediální data obsahují většinou různé zkreslené, nepodstatné informace (šum), je očividné, že není potřeba počítat SVD rozklad na celé matici. Právě proto aplikujeme metodu Monte-Carlo, pomocí níž snížíme velikost matice. Poté je matice zpracována metodou LSI. Konkrétně jsou zpracovány obrázky, které byly pořízeny z bezpečnostních kamer. Prezentované výsledky podobností jsou velmi příznivé.

Klíčová slova: LSI, SVD,Metoda Monte Carlo, matice, vektor

Abstract

This bachelor thesis deal with effective application Monte-Carlo Method over SVD decomposition. SVD decomposition is an effective and robust decomposotion method of linear algebra. The application of this decomposition is in all computers sciences. For example, compression of pictures, the SVD is used in the PCA method, to analysis for ex. to human faces. Common computing of SVD is demanding to CPU and RAM, primarily in case of very large matrix. But multimedia data contains often unimportant information, so we dont have to compute an approximation of the SVD of complete matrix. We applicate Monte-Carlo method, with this method we sucessfully reduce the size of the matrix.

Keywords: LSI, SVD,Method of Monte-Carlo, matrix, vector

Seznam použitých zkratek a symbolů

LSI	–	Latentní sémantické indexování
SVD	–	Singular value decompostion-singulární rozklad matice A
MC	–	Metoda Monte-Carlo
k	–	počet prvních σ_k singulárních čísel matice S
A	–	Matice A
S	–	Matice singulárních čísel

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Původně bylo zadaní: aplikace a testování na textových dokumentech včetně LSI metody, já ji modifikoval o aplikaci na obrazová data. Obrázky zachytávají pohyb, jedná se o bezpečnostní kamery, s venkovním i vnitřním použitím. Obrázky jsou prostorové, identifikační. V barevné (RGB) i černobílé podobě. Zdroj obrázků je již zmíněný bezpečnostní dohledový systém. Vlastník souhlasil se zveřejněním, i všechny zúčastněné osoby, kterých by se to mohlo týkat, vydaly souhlas se zveřejněním. Abychom otestovali co nejefektivněji SVD rozklad metodou Monte-Carlo, musíme analyzovat co možná nejvíce rozsáhlá data, jak po stránce kvalitativní, tak po stránce kvantitativní.

Obrazová data obsahují mnoho skrytých vazeb a naopak informací, které jsou nepodstatné, proto jsou ideální pro otestování. Abychom měli co nejlepší zpětnou vazbu, pro testovaná data, použijeme metodu LSI-Latentní sémantické indexování, na dotazování dokumentů (obrázků) a vyhledávání, třídění k nim nejpodobnějším. Tato metoda převede všechny dokumenty do vektorů a sestaví z nich matici dokumentů. Díky tomuto přístupu můžeme aplikovat na tuto výslednou matici SVD rozklad. SVD rozklad slouží ke snížení dimenze matice dokumentů, díky tomuto se zlepšují výsledky, v důsledku odhalení skrytých a podstatných vazeb mezi dokumenty. Navíc SVD rozklad můžeme úspěšně aproximovat metodou Monte-Carlo k získání rozměrově menší, ale zároveň informačně výhodnější matice.

Byly přidány paralelní implementace k původní implementaci MC na SVD. Celý tento celek se obecně nazývá CBIR systém (Content based-image retrieval) [17], výchozí obrázek tzv. Query obrázek se nastaví a podle obsahu obrazových dat se zobrazí všechny jemu nejpodobnější. V první kapitole je definován SVD rozklad, v další kapitole seznámení se z základním principem LSI metody a ve 4. Kapitole úvod do Monte-Carlo metody, aplikace MC metody na částečný SVD rozklad je napsána v 5. kapitole, v 6.kapitole následně jsou prováděny experimenty na obrazových datech.

2 SVD rozklad

2.1 Úvod

Tato část má za úkol nás seznámit s teoretickým základem SVD Rozkladu.[2, 3].

2.1.1 Definice

Definice 2.1 *Máme danou matici $A \in R^{m\times n}$, SVD rozklad je definován jako součin tří matic:*

$$A = U^T SV. \tag{1}$$

kde $V \in R^{n\times n}$ je ortogonální matice s vlastními vektory a $U \in R^{m\times m}$ je ortogonální matice s vlastními vektory, a S je diagonální matice, na jejíž diagonále jsou vlastní čísla matice $\sqrt{A^T A}$ pro $m \geq n$, nebo matice $\sqrt{AA^T}$ pro $m \leq n$ s nezápornými reálnými prvky, které jsou uspořádány v sestupném pořadí, kde vztah pro diagonální prvky je dán takto:

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq ..... \geq \sigma_{min(m,n)}. \tag{2}$$

Čísla σ_i jsou singulárními čísly matice A, a sloupce matice U,V jsou levými, pravými singulárními vektory matice A, jestliže pracujeme pouze s případem kdy $m \geq n$, jestliže $\sigma_1 \neq 0$ a zároveň $\sigma_1 \geq \sigma_{b+1} = \ldots = \sigma_n = 0$. Potom b je hodnost matice A a obecný SVD rozklad můžeme zapsat jako:

$$\begin{pmatrix} a_{1,1} & \ldots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \ldots & a_{m,n} \end{pmatrix} = \begin{pmatrix} u_{1,1} & \ldots & u_{1,b} \\ \vdots & \ddots & \vdots \\ u_{m,1} & \ldots & u_{m,b} \end{pmatrix} \cdot \begin{pmatrix} \sigma_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_b \end{pmatrix} \cdot \begin{pmatrix} v_{1,1} & \ldots & v_{n,1} \\ \vdots & \ddots & \vdots \\ v_{1,b} & \ldots & v_{n,b} \end{pmatrix} \tag{3}$$

Příklad 2.1
Příklad SVD rozkladu na obdélníkovou matici 3×4 :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} = \begin{pmatrix} -0.2067 & -0.8892 \\ -0.5183 & -0.2544 \\ -0.8298 & 0.3804 \end{pmatrix} \begin{pmatrix} 25.4368 & 0 \\ 0 & 1.7226 \end{pmatrix} \begin{pmatrix} -0.4036 & 0.7329 \\ -0.4647 & 0.2898 \\ -0.5259 & -0.1532 \\ -0.5870 & -0.5962 \end{pmatrix}$$

■

2.2 Normy vektorů a matic

Nadefinujeme si zde pouze Euklidovskou normu a Frobeniovu normu matice.

2.2.1 Euklidovská norma

Definice 2.2 *Na prostoru R^n lze definovat tzv. euklidovskou normu vektoru $x = (x_1, x_2, ..., x_n)$ jako*

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 \cdots + x_n^2} \tag{5}$$

2.2.2 Frobeniova norma

Definice 2.3 *Nechť $A \in R^{m \times n}$ pak*

$$\|\mathbf{A_F}\| = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} \|\mathbf{A_{ij}}\|^2} \tag{6}$$

je Frobeniova norma matice A.

3 LSI

3.1 Úvod

Tato část má za úkol nás seznámit s teoretickým základem LSI .[4] [12] [13]

3.1.1 Definice

Latentní sémantické indexováni je vyhledávací a indexovací metoda, primárně je určena a byla zkonstruována pro analyzování textových dokumentů, je v podstatě založena na principu, že slova používaná ve shodných souvislostech konvergují ke stejnému významu. LSI je tedy schopna separovat smysluplný obsah analyzovaného textu pomocí základních asociací mezi těmito pojmy, které existují v podobném kontextu. [19]

V praktickém využití mnohdy využívá metody lineární algebry, hlavně SVD rozklad, který slouží k snížení dimenze prostoru dokumentů, konkrétně k zachování prvních σ prvků matice S, to znamená, že dojde k vypuštění nejméně podstatných informací.

Slova mají mnoho významů, synonym, proto naopak dojde ke zpřesnění výsledků, díky odhalení skrytých (latentních) vazeb mezi podstatnými prvky (pojmy) a dokumenty. např. konkrétně v našem případě obrazových matic se jedná o skrytě podobné prvky všech obrazových matic v matici dokumentů A.

3.1.2 Sestaveni matice dokumentů

Pomocí tak zvané vektorizace, obrázek 1, jednoduše sestrojíme matici dokumentů A, kde je každý dokument uspořádán jako vektor ve dvojrozměrném prostoru, díky tomuto přístupu je možné aplikovat nástroje pomocí lineární algebry, jako je např. již zmiňovaný SVD rozklad.

Matice dokumentů obsahuje kódované rastrované obrázky, v našem případě má $m*n$ řádků, kde m, n, jsou rozměry obrázků, tj. např. $i = m * n = 320 * 240 = 72600$, matice A tedy obsahuje i řádků a j sloupců-počet obrázků.

3.1.3 Výhody metody LSI

LSI řeší nejčastější problémy, v případě analyzování textových dokumentů, již zmíněných synonym a slov mnohovýznamových. [15]

LSI se dá s výhodou použít i ke kategorizaci dokumentů, když vytvoříme předem definovanou matici dokumentů, kde jednotlivé dokumenty tvoří jednotlivé kategorie, a jako dotaz zadáme zkoumaný dokument. Ve výsledku, zpravidla se určuje kategorie podle prvního, tedy nejbližšího výsledku od dotazu, případně se výsledky můžou kombinovat v závislosti na naprogramovaném postupu rozhodování.

Další důležitou výhodou je skutečnost, že díky matematickému aparátu je LSI absolutně imunní vůči jazyku dokumentů nebo jejich kombinací. Použití ve zpracování textových dokumentů jako jsou např.: Slovníky, jazykové překlady, učebnice. [18]

Nicméně Metoda LSI není všemohoucí: je třeba si uvědomit, že všechny analyzované dokumenty musí mít stejnou dimenzi. Nastává mnoho situací v praxi, kdy jednotlivé

dokumenty mají individuální obsah stěžejních, kvalitních informací potřebných pro co nejpřesnější výsledky. Tento menší problém více méně řeší SVD rozklad. Proto se aplikují různé optimalizační úlohy až na výslednou matici připravenou na SVD rozklad.

3.1.4 Dotazování dokumentů

Uživatel zadává dotaz ve stejné podobě, jako vektor (v našem případě vektorizovaný obrázek), na výstupu očekává množinu odpovídajících vektorů (dokumentů). Dotaz uživatele musí být předzpracován, musí tedy projít syntaktickou analýzou, odstraněním frekventovaných informací, lemmatizací atd. nebo např. aplikování optimalizačních úloh, jako je např. Monte-Carlo metoda.

Dotaz je tedy opět dokument, pro který hledáme dokumenty relevantní k dotazu uživatele.

Podobnost ve vektorovém modelu chápejme jako vzdálenost jednotlivých vektorů v matici dokumentů A.

Ve výsledku dostaneme na výstupu setříděné relevantní dokumenty dle jejich vzdálenosti od dotazovacího vektoru. Z provedených experimentů bylo zjištěno, že poměrně nejlépe charakterizuje podobnost vektorů v matici dokumentů A kosinová podobnost, tedy úhel dvou vektorů vyjádřený pomocí skalárního součinu vektorů u a v.

$$\cos(\theta) = \frac{(u, v)}{\sqrt{(u, u)}\sqrt{(v, v)}} \tag{7}$$

Upravíme pro potřeby LSI [4]

$$\cos(\theta) = \frac{(q, D)}{\sqrt{(q, q)}\sqrt{(D, D)}} \tag{8}$$

q je označení pro query dokument, D jako zbytek porovnávaných dokumentů

$$\cos(\theta) = \frac{((A_k e_j)^T q)}{||(A_k e_j)||_2 ||q||_2} \tag{9}$$

kde e_j označuje j-tý kanonický vektor dimenze j tz. j-tý sloupec jednotkové matice $n \times n$ I_n. Sloupcový vektor $A_k e_j$ je tedy j-tý sloupec matice A_k s hodností k. Výraz můžeme dále upravit na

$$\cos(\theta) = \frac{((U_k \sum_k {V_k}^T e_j)^T q)}{||U_k \sum_k {V_k}^T e_j||_2 ||q||_2} \tag{10}$$

dále platí

$$(U_k \sum\nolimits_k {V_k}^T e_j)^T = ({e_j}^T (V_k^T)^T \sum\nolimits_k^T {U_k}^T) = \tag{11}$$

$(A^T)^T = A$, pro diagonální čtvercovou matici D platí :$(D^T) = D$

$$= ({e_j}^T V_k \sum\nolimits_k {U_k}^T) \tag{12}$$

Po dosazení do rovnice

$$\cos(\theta) = \frac{{e_j}^T V_k \sum_k ({U_k}^T q)}{||{e_j}^T V_k \sum_k ||_2 ||{U_k}^T q||_2} \tag{13}$$

Označme s_j k-redukovaný vektor dokumentu j, $s_j = {e_j}^T V_k \sum_k$, pak můžeme psát

$$\cos(\theta) = \frac{s_j({U_k}^T q)}{||s_j||_2 ||{U_k}^T q||_2} \tag{14}$$

Měříme tedy kosinovou podobnost mezi vektorem $U_k^T q$, což je promítnutí dotazovacího vektoru q do sloupcového prostoru A_k, hledáme tedy souřadnice dotazovacího vektoru ve sloupcového prostoru A_k s bázi U_k, a n k-redukovanými vektory dokumentů.

Upravený vzorec pro potřeby LSI.

3.1.5 Úprava SVD rozkladu na částečný SVD rozklad(VD rozklad)

Díky dobře známým vztahům mezi SVD rozkladem na matici$A$ aVD rozkladem na matici $A^T A$ z lineární algebry [5], můžeme postupně odvodit:

$$A = USV^T \tag{15}$$

$$A^T = (USV^T)^T = VS^T U^T \tag{16}$$

$$A^T A = VS^T (U^T U) SV^T = VS^T SV^T \tag{17}$$

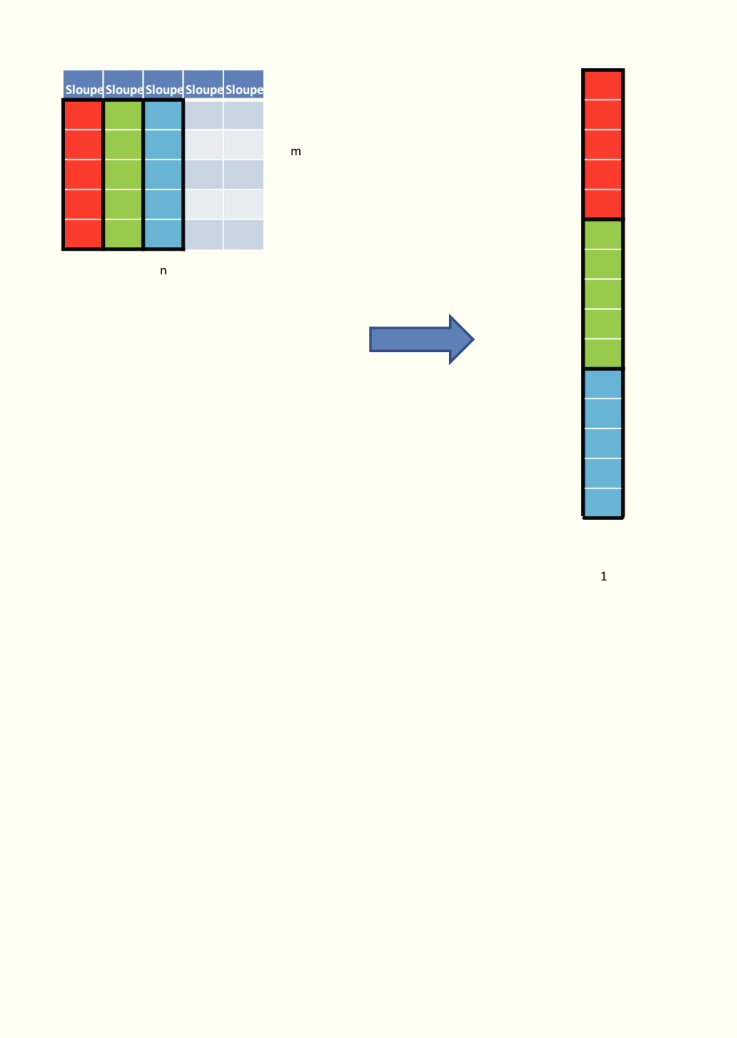
Protože víme, že matice V je ortogonální, následující rovnost je dána vztahem

$$AV = US \tag{18}$$

$$\mathrm{AVS}^+ \approx U \tag{19}$$

kde symbol S^+ je označení pro Moore-Penrose pseudoinverzi (pinv).

U součinu $A^T A$ můžeme využít toho, že výsledná matice není jenom čtvercová, ale i symetrická. Díky tomu nemusíme při maticovém násobení počítat všechny součiny matice, ale jen, co jsou např. pod diagonálou a samozřejmě na diagonále, zbytek prvků překopírujeme zrcadlově.



Obrázek 1: Ukázka vektorizace dokumentů, tento postup se postupně aplikuje na všechny dokumenty, pomocí nichž se sestaví matice dokumentů A.

4 Metoda Monte-Carlo

4.1 Úvod

Tato část má za úkol nás seznámit s teoretickým základem metody Monte-Carlo.[1]

4.1.1 Definice

Numerická výpočetní Metoda Monte-Carlo patří mezi stochastické metody, poskytuje řešení v podobě aproximace různých matematických problémů, jejichž podstata je založena na statistickém experimentálním vzorkování, jde v podstatě o Matematický model (systém) reálného problému (děje) (Analogový model), který závisí nějakým způsobem na opakování náhodném výběrů vzorků (odhadech náhodné veličiny), kdy po určitém počtu opakování, můžeme výběr zpracovávat klasickými statickými metodami, jako jsou například: průměr a směrodatná odchylka, modus, medián, atd. Přesnost metody MC je tedy závislá na počtu opakovaní náhodného děje. Metoda Monte-Carlo nepatří mezi nejefektivnější, protože rychlost konvergence chyby výsledku k nulové hodnotě je u Metody Monte-Carlo přibližně rovna převrácené hodnotě odmocniny z počtu realizovaných experimentů.

Metoda Monte-Carlo je poměrně mladá metoda nacházející uplatnění v mnoha oborech: od ekonomie, biologii, po IT, matematiku, hazardní hry, simulacích přírodních katastrof a nebo katastrof vytvořených uměle lidským faktorem, jako je např. vodíková, atomová bomba.

Analogový model aplikujeme tehdy, když simulujeme určitý skutečný děj. Kde známe všechny pravděpodobnostní rozložení všech náhodných veličin.

Neanalogový model, taky nazývaný geometrický model metody MC, se používá když nepoužíváme model reálného děje, např. při výpočtu určitého integrálu nebo např. obsahu ohraničeného útvaru.

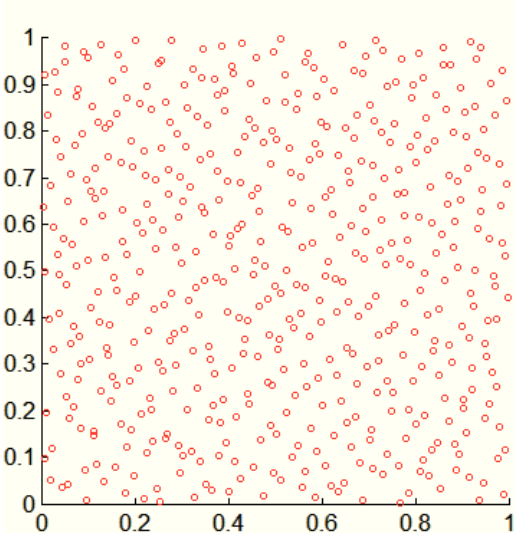
4.1.2 Generování náhodných čísel

[8] [10] [11]

Generování náhodných čísel je stěžejní pro přesnost Metody Monte-Carlo. Generátory náhodných čísel, přesněji řečeno generátory pseudo-náhodných čísel, můžeme rozdělit pomocí dvou odlišných přístupů generování náhodných čísel:

- Lineární kongruentní generátor.
- Quasi-nahodné čísla.

Quasi-náhodná čísla fungují na principu slabých neshod jednotlivých sekvencí, tzn. že generovaná čísla se snaží co nejvíce zaplnit prostor tak, aby byla co nejrovnoměrněji rozložena, kde každý prvek, většinou počáteční prvek zase vychází z HW šumu, který se přepočítává pomocí speciálních polynomů, záleží na volbě mezi nejznámějšími Haltonovými sekvencemi nebo Sobolových sekvencí.



Obrázek 2: Quassi-náhodná čísla, generováno 500 náhodných čísel v rozsahu $[0, 1]$.

Použití Quasi-náhodných čísel přineslo sice jistou stabilitu výsledků, nicméně je zase ochuzena o prvek ”překvapení”, kdy i při opakování náhodných vybraných čísel, což u Quasi je téměř nemožné, viz (obr. 2), docházelo k příznivějším výsledkům, navíc samozřejmě generování Quasi náhodných čísel je časově náročnější. Zpravidla se nastavuje s přeskokem prvních $100-1000$ čísel s nastavením generování v troj-rozměrném prostoru.

Nechť $I^s = [0, 1]$ je s-dimenzionální jednotka hyperkrychle a f je reálná integrovatelná funkce na I^s. Původní motivace Sobol-a bylo vytvořit sekvenci x_n v prostoru I^s tak, že:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) = \int_{I^s} f \tag{20}$$

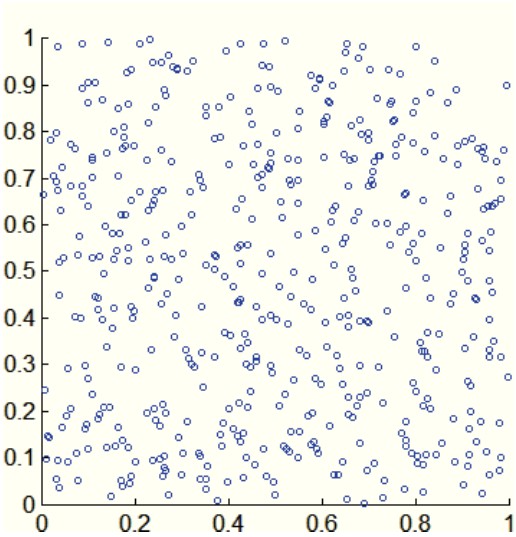
jen tak rychle , jak je to jen možné.

V této práci se budeme bavit pouze o lineárním kongruentním generátoru. Ten je dán obecným vztahem:

$$x_{i+1} \quad = \quad (a * x_i + c) \bmod m \tag{21}$$

konstanty a, c a m musejí být vhodně zvolené. Počáteční nastavení x_0 neboli random seed, zase vychází z HW šumu. Generátor generuje celá čísla s rovnoměrným rozložením v rozsahu $0 \leq x_i \leq m$.

Po určité době generování náhodných čísel může nastat nežádoucí periodicita generování náhodných čísel. Ještě větší problém nastává při nesprávném zadání hodnot, a, c a m, kdy dojde k určité linearitě generování náhodných čísel. Tyto problémy jsou částečně vyřešeny v `Matlabu` speciální funkcí, kde dochází k resetování do různých stavů v každém čase(v každém novém zavolání):



Obrázek 3: Pseudonáhodná čísla, generováno 500 náhodných čísel v rozsahu $[0, 1]$.

```
O'Clock.
rand('state',sum(100*clock));
```

Výpis 1: Clock Matlab

4.2 Obecný postup aplikace metody Monte-Carlo

Obecně metodu Monte-Carlo aplikujeme při řešení problému tehdy, když chovaní celkového systému ovlivňuje náhodný prvek, tedy buď příčinu či vysvětlení jevu neumíme nalézt, racionálně vysvětlit nebo jev žádnou příčinu nemá.

Obecný princip metody Monte-Carlo se zdá být jednoduchý, nicméně nalezení efektivní aplikace na reálný problém je v mnoha případech poněkud těžší. Obecný postup aplikace ale můžeme rozdělit jako:

- Návrh řešení, analyzování problému, zohlednění všech veličin, které ovlivňují a mohou ovlivnit chování celkového systému.
- Generování náhodných čísel, výběr mezi principy generování náhodných čísel: Uniformita, stabilita vs. moment překvapení, opakování náhodných čísel.
- Přeměna jednotlivých náhodných veličin na určité pravděpodobnostní rozdělení.
- Provádění následujících experimentů s (většinou) pevně zadaným počtem opakování.
- Statistické analyzování výsledků - hledaná hodnota je zpravidla dána některým z momentů statistických veličin, nejčastěji střední hodnotou. A následně úprava

některých proměnných, které ovlivňují systém a následné provádění dalších experimentů, k nalezení co nejoptimálnějších vstupních nastavení.

Příklad 4.1
Příklad geometrického modelu Metody Monte-Carlo, na výpočet čísla π :

Máme daný čtverec, kterému je vepsaná čtvrtkružnice. Je možné jeho hodnotu zjistit náhodným házením špendlíkovou hlavičkou, popřípadě jakýmkoliv jinými drobnými předměty do prostoru čtverce, a výsledný poměr počtu všech hodů a hodů do kruhu dá hodnotu čísla π. A nebo také, můžeme generovat čísla v rozsahu rozměrů čtverce.

Nadefinujeme si obsah čtvrtkružnice, obsah čtverce:

$$S_1 = \frac{\pi r^2}{4} S_2 = r^2 \tag{22}$$

jejich poměr je pak:

$$\frac{S_1}{S_2} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} \tag{23}$$

Výjádření Ludolfova čísla π :

$$\pi = 4\frac{S_1}{S_2} \tag{24}$$

■

5 Aproximace SVD rozkladu s využitím Metody Monte-Carlo

Princip aplikace Monte-Carlo metody na SVD rozklad je následující.

Simulační SVD algoritmus vybere náhodně předem daný počet řádků pomoci normy vektorů (řádků), s větší pravděpodobností pomocí náhodného výběru vybere ty, které mají větší změnu, to jest ty, které mají větší vypovídající hodnotu, následně se vypočte SVD rozklad. Viz okomentovaný kód. Kód byl převzat [6] a mírně upraven.

Byly přidány parfor cykly, pro paralelní výpočty, řádově až $50 - 60$ krát rychlejší výpočty norem. Co se týká maticového násobení $A^T A$ tak v matlabu jsou speciální knihovny, které rozpoznají symetrickou matici, tudíž aplikování mého modifikovaného maticového násobení v matlabu by bylo zbytečné.

Přepsán byl ale do Javy a jazyka c++, viz příloha. Vstupní matice A v našem konkrétním případě obsahují vektorizované obrázky, jež představují jednotlivé sloupce matice A.

```
NORMAFRO = norm(A,'fro')^2;
```

Výpis 2: Frobeniova norma vstupní matice

```
co = s/NORMAFRO;
```

Výpis 3: Co obsahuje jakou má hodnotu průměrně 1 vybraný řádek Frobeniovy normy

```
rand('state',sum(100*clock));
```

Výpis 4: Definování funkce random

```
parfor i=1:m,
 ER(i) = norm(A(i,:))^2;
end
```

Výpis 5: Paralelní for pro rychlejší výpočet načtení všech norem řádků do pomocné proměnné ER

Následuje vytvoření pomocné proměnné, díky níž dostaneme setříděné pole a zároveň pro velké změny norem dostaneme velké rozdíly hodnot v pomocné proměnné PROB, tedy velkou pravděpodobnost výběru této normy řádku, tedy tohoto konkrétního řádku.

```
PROB(1) = ER(1);
for i=2:m,
 PROB(i) = PROB(i-1) + ER(i);
end
```

Výpis 6: Načtení všech norem řádků z proměnné ER do předchozích i kroků pomocné proměnné PROB pro lepší detekci velkých změn norem.

```
parfor i=1:s,
  ROW = ceil (rand * NORMAFRO);
  left = 1;
  right = m;
```

```
  while abs(left−right) > 1
   if ROW > PROB(ceil((left+right)/2))
       left = ceil (( left+right )/2);
   else   right = ceil (( left+right )/2);
   end
  end
  L(i ,:) = A( left ,:) /( sqrt(co∗ER(left)));
end
```

Výpis 7: Modifikované binární vyhledávaní nejbližších vektorů k náhodně vybranému číslu

```
  L(i ,:) = A( left ,:) /( sqrt(co∗ER(left)));
end
```

Výpis 8: Každý vybraný řádek se podělí z důvodu zachování stejné Frobeniovy normy-stejné energie matice

Vstupní transponovanou matici A vynásobíme maticí A. Následně se provede na výsledné matici částečný SVD rozklad. Matici S s vlastními čisly musíme odmocnit. Následuje výpočet matice U.

```
AA=A'∗A;

[V,S]=eigs(AA,k);

S=sqrt(S);

U = A ∗ V ∗ pinv(S);
```

Výpis 9: Násobení matic-VD Rozklad-odmocnění matice S-Matice U

Výpočet dotazovacího vektoru q_c.

```
q_c=(q'∗U)∗pinv(S);

norm_{qc}=norm(q_c);
```

Výpis 10: Dotazovaný dokument-norma vektoru

```
for i=1:size(A,2) % loop over all documents

   % Computes the similarity coefficient for i−th document
   doc_c(i) = ( q_c∗V(i ,:) ')/( norm_q_c∗norm(V(i,:)));



end;
```

Výpis 11: Výpočet kosinové podobnosti

5.1 Obrázky a tabulky

Testování probíhalo v programu `Matlab` s využitím naprogramovaného LSI pro `Matlab` od pana Ing. Pavla Prakse, Ph.D [5] modifikováno mnou o metodu MC, na konečném počtu 1100 obrázků z pěti bezpečnostních kamer, obrázky byly zcela náhodně vybírány, převedeny byly do černobílé barvy,použito bylo rozlišení $160 * 120$, matice je hustá (dense matrix), kvalita záznamu z bezpečnostních kamer není moc dobrá, tím překvapivější jsou dobré výsledky s pomocí metody Monte-Carlo. Většina z kamer je nastavena na monitorovací úroveň, jedna z nich na identifikační úroveň. Z mých pokusů vyplynulo, že nejlépe (nezávisle na zkoumaném výchozím obrázku) funguje MC s 51-60 procenty řádků. U vyššího procentuálního výběru řádků nejsou výsledky znatelné ani v rámci úspory času, paměti, níže zase nastává značná nestabilita výsledků a špatný odhad k parametru v matici S. Pomocí parametru k nastavujeme počet k prvních singulárních čísel matice S, čím menší je parametr k, tím dochází ke zpřesnění výsledků a zároveň k rychlejšímu výpočtu, nastavení tohoto parametru není jednoduché, protože příliš nízké k zase způsobuje velkou ztrátu zájmových informací, musí se odhadnout, nejlépe se nastaví před viditelnou velkou změnou singulárních čísel, např. v obr. 19 se k nastaví jako 20. Zde je zobrazeno pár výsledků, kdy metoda funguje dobře a kdy ne. V drtivé většině fungovala MC velmi dobře, nicméně je vidět, že i u zobrazeném obrázku 8 obrázky kolem minus 0.3 nemají přímou souvislost s query (člověk na schodech). Obrázky blízké mínus 1 (např.-0.8) jsou si opět podobnější s query – negativní korelace. Co se týká času výpočtů vlastních čísel a vektorů viz tabulka 1, tak metoda MC fungovala zhruba vždy o 1 sekundu lépe, nicméně výpočet metody MC trval zhruba stejně dlouho tj. 1 sekundu. Všimněme si zajímavé skutečnosti, co se týká větších objektů-aut, jsou seřazeny v drtivé většině ve stejném směru jako je dotazovaný dokument (obrázek), to je velice cenná vlastnost při bezpečnostních kamerových dohledech, pro kvalitnější identifikaci.

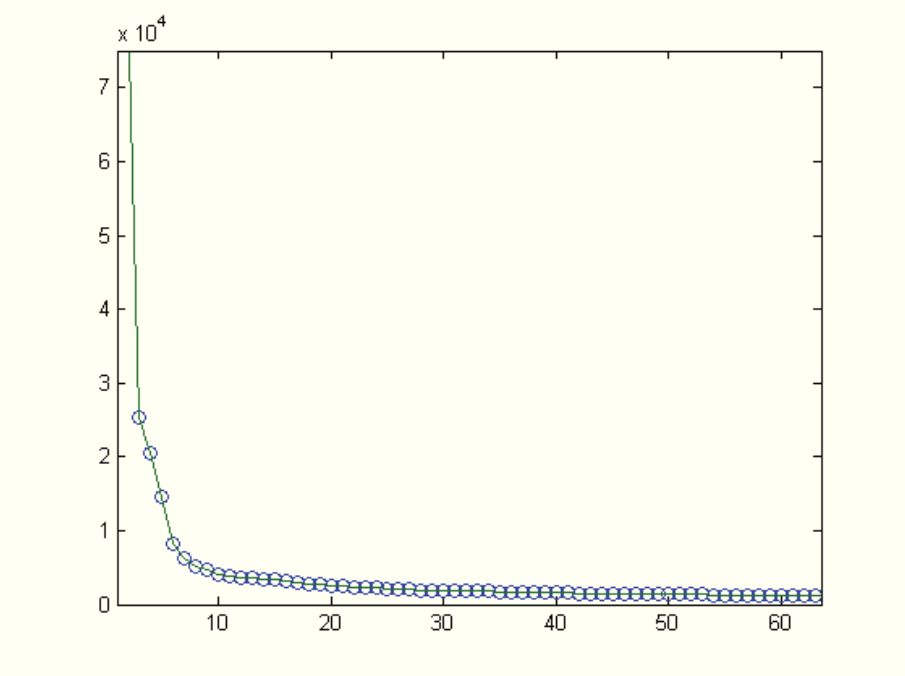
5.2 Tabulka změřených hodnot

- CPU Time - měření výpočtů vlastních čísel a vlastních vektorů (v sekundách)
- k - hodnota-první σ_k singulární čísla matice S
- s - procentuální počet řádků vybraný metodou MC z původní matice
- Využití paměti RAM (MC) - procentuální využití paměti RAM s pomocí metody MC vzhledem k původní matici bez MC

Obr.	k	CPU Time (MC)	% s	Využití paměti RAM(MC)%	CPU Time
4	9	0.1094	56	85	1.1094
6	20	0.3750	56	85	1.4526
7	20	0.198	56	85	1.0256
8	20	0.1313	56	85	1.04
9	9	0.0142	20	42	0.94
11	10	0.17	56	85	1.11
12	6	0.192	30	66	1.252
13	11	0.141	56	85	1.413
15	13	0.421	56	85	1.101
16	20	9.421	56	80	11.101
17	20	10.00	56	80	10.44

Tabulka 1: Tabulka naměřených hodnot, testování probíhalo na PC Intel 2.93Ghz Dual Core, s pamětí RAM 6GB, Operační systém Windows Xp 32 bit

Obrázek 4: Query obrázek auto, při k=9 a s=56 procent, vynikající výsledek, díky optimálnímu nastavení s hodnoty a k hodnoty. Zde se ukázala síla MC metody. Všimněme si, že auta jsou seřazena většinou ve stejném směru jízdy.



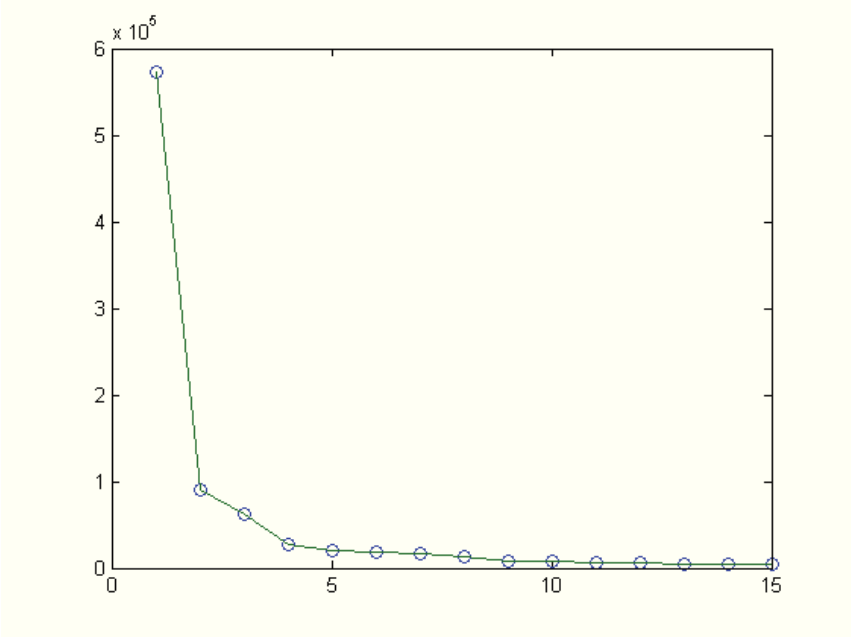
Obrázek 5: Graf singulárních čísel Matice S. Zde vidíme případ, že hodnota k=9 funguje velmi dobře, i když náhlá změna velikosti k začíná už na hodnotě 25-30.

Obrázek 6: Query obrázek auto, při k=20 a s=56 procent, zde vidíme velmi dobrý výsledek, v důsledku optimálně nastavené k, s, hodnoty.

Obrázek 7: Query obrázek auto, při k=20 a s=56 procent, zde je podobnost blízká nule. Podobnost blízké nule mají objekty, které nemají přímou souvislost s query - např. traktor či cyklista.

Obrázek 8: Query obrázek auto, při k=20 a s=56 procent, zde je příklad výsledků, kdy je kosinová podobnost blízká mínus 1. Jsou zde zobrazeni lidé a na konci překvapivě také auta, ale v opačném směru! Modré auto před nimi je rozostřené, nejsou vidět zájmové objekty auta, z těchto důvodů máme zobrazeny tyto obrázky na konci.

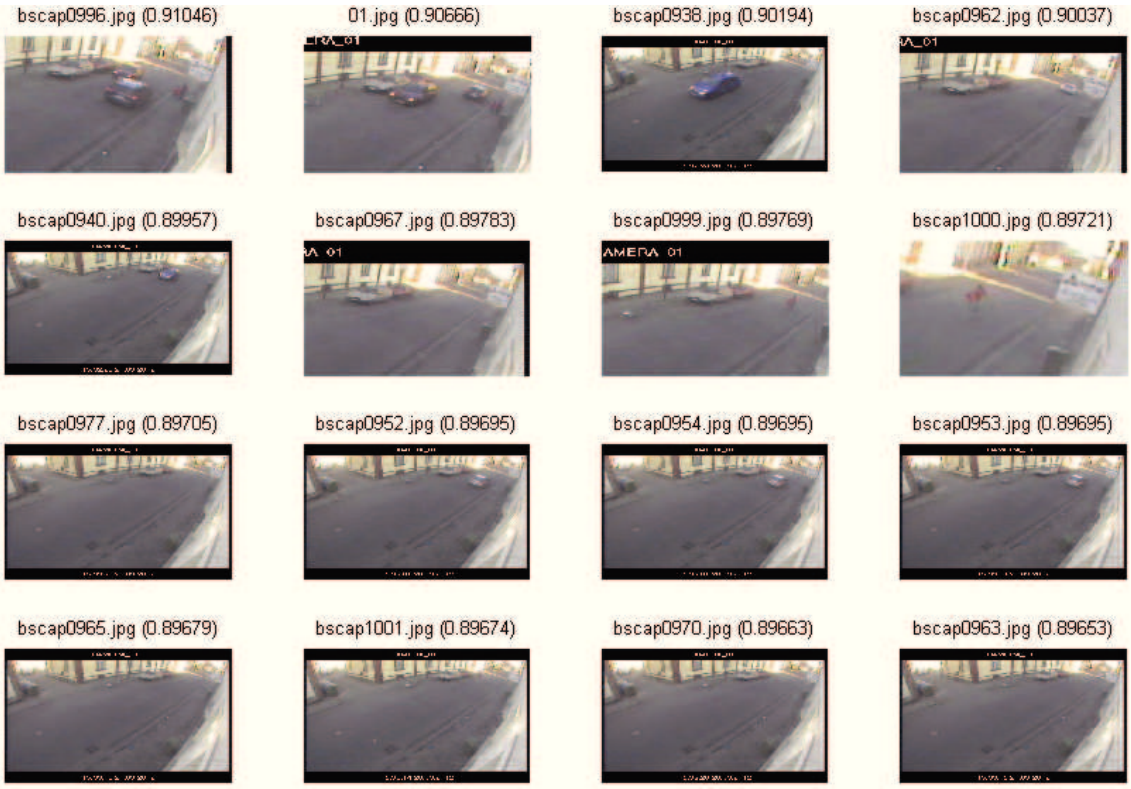
Obrázek 9: Query obrázek auto, při k=9 a s=20 procent, zde vidíme ne dobrý výsledek, způsobený drastickým snížením s, vybraných 20 procenty řádků, včetně malé hodnoty k, díky tomu dojde k vypuštění zájmových objektů a zobrazí se pouze pozadí Query obrázku.



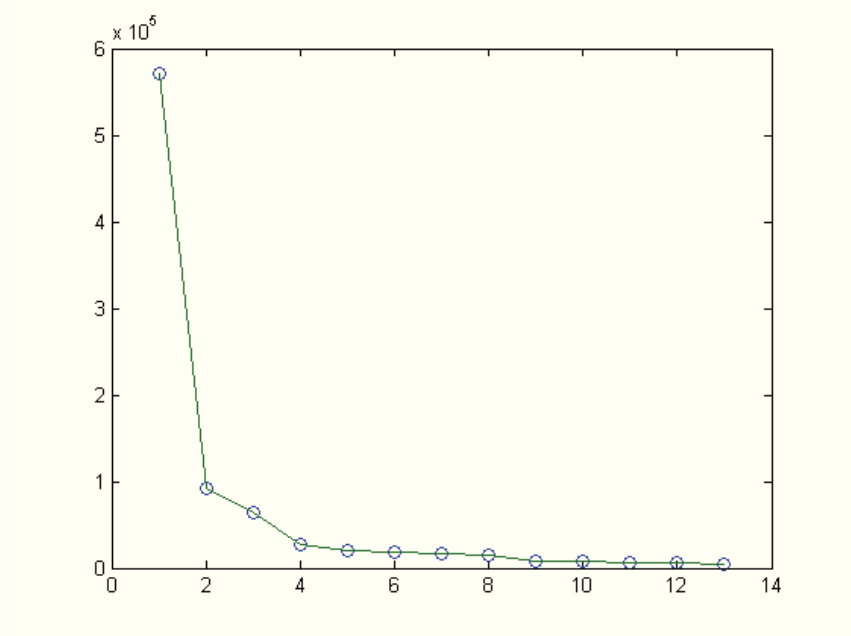
Obrázek 10: Graf s=20, k=9, vidíme zde špatně zjistitelné odhadnutí náhle změny singulárních čísel.

Obrázek 11: Query obrázek člověk, při k=10 a s=56 procent, zde vidíme velmi dobrý výsledek způsobený optimální hodnotou s a k, člověk je zobrazen i z jiné kamery.

Obrázek 12: Query obrázek člověk, při k=6 a s=30 procent, výsledek není dobrý, je způsobený velmi sníženou hodnotou s a malým k hodnoty matice S SVD rozkladu, v důsledku toho dojde nejdříve k vypuštění zájmových informací a následně důležitých singulárních čísel, proto taky je seřazeno po lidech hned pozadí.



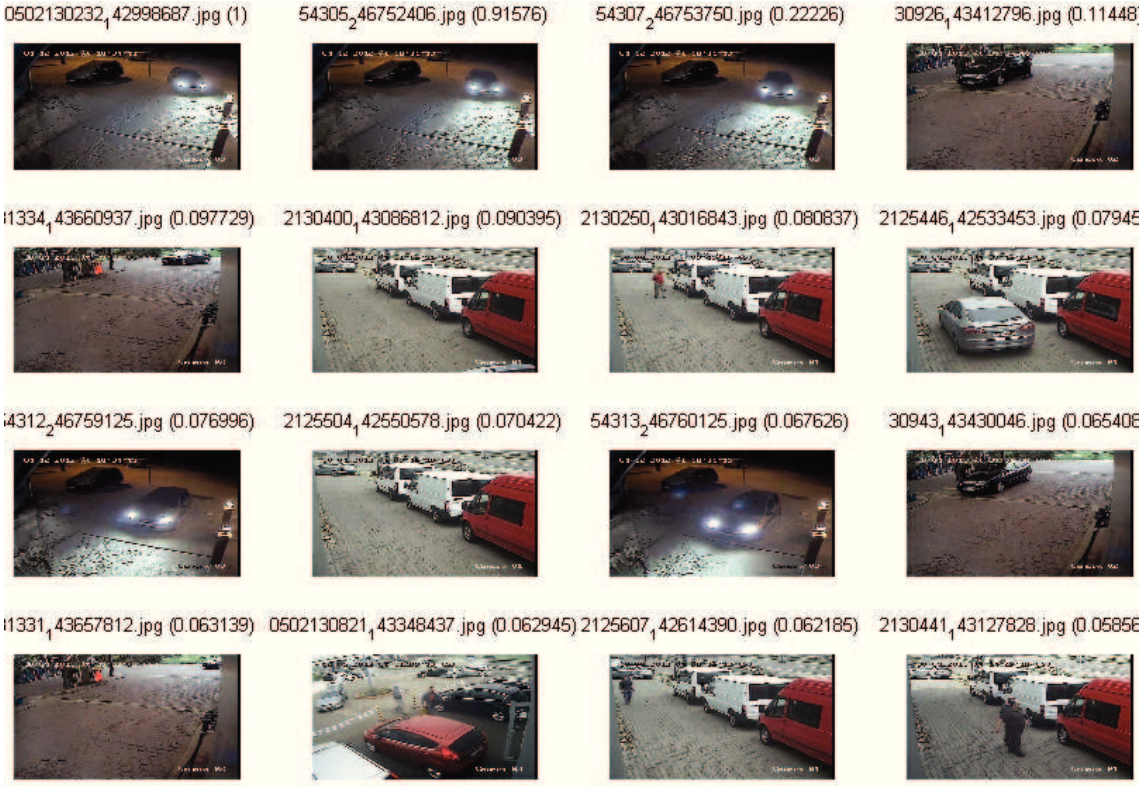
Obrázek 13: Zde je testován těžší Query obrázek auto plus člověk, při k=11 a s=56 procent, výsledek vynikající, pravděpodobnost, že se setkají dvě auta zároveň a zároveň člověk je velmi malá, proto je v databázi počet takových obrázků velmi malý, proto tak vidíme častěji nalezení buď auta nebo člověka samostatně.



Obrázek 14: Graf zde je testován těžší Query obrázek auto plus člověk, při k=13 a s=56 procent.

Obrázek 15: Zde je testován těžší Query obrázek auto plus člověk, při k=13 a s=56 procent, výsledek dobrý, nicméně lepších výsledků se dosahuje ještě při snížení k hodnoty viz obrázek 14.

Obrázek 16: RGB barva, query auto v červené barvě, seřazeny auta v červené barvě z jiné kamery, zajímavost: zároveň jsou zobrazeni i lidé-také v červené barvě, s=56 procent, k=20.



Obrázek 17: HSV barevný systém, testován těžký obrázek, auto- v noci , navíc s rožnutými světly, zároveň tmavé auto v pozadí, hodně ztěžující okolnost, výsledek je nicméně dobrý, s=56 procent, k=20.

6 Závěr

Cílem bakalářské práce bylo implementovat a numericky otestovat simulační algoritmus SVD s využitím stávajících knihoven SVD. Tato bakalářská práce využívá výsledků simulačních algoritmů, viz ([6]). Oproti předchozímu výzkumu, který byl aplikován na analýzu textových dat, je těžiště této práce věnováno vyhledávání v obrazových datech. Analyzován je i vliv barevného kódování obrazu na vyhledávání podobnosti. Pro implementaci byl využit potenciál Matlabu, včetně paralelní implementace simulačního algoritmu, viz ([6]). Funkčnost vytvořeného simulačního algoritmu byla úspěšně demonstrována na algoritmu metody Latent Semantic Indexing, která byla využita pro automatické vyhledávání obrázků. V rámci bakalářské práce byla pro tento účel vytvořena vlastní testovací obrazová databáze, která obsahuje zejména reálné digitální obrazy dopravních prostředků. Obrazová data byla snímána pevně umístěnými digitálními monitorovacími kamerami. Záznam zvuku nebyl prováděn. Simulačním algoritmem bylo provedeno testování barevných i černobílých obrazů. V bakalářské práci byly analyzovány reálné obrazové data. S cílem minimalizovat vliv kolísavých světelných podmínek a dalších faktorů, které ovlivňovaly výsledné fotografie, např. vliv počasí, byla obrazová data transformována do černobílé podoby. Úspěšnost testování algoritmu byla 90 procent. Tedy pouze v 10% případů byl mezi nejbližšími výsledky ne zcela správně klasifikovaný objekt. Byly taktéž testovány různé systémy pro kódování barevných digitálních obrazů: `RGB`, `HSL`, `HSV`, `YIQ`. Numerické experimenty zatím neprokázaly žádné zkvalitnění výsledků, úspěšnost vyhledávání zůstala na 90 procentech.

Samozřejmě, pro detailnější zobrazení podobnosti by bylo výhodné použít pro vyhodnocení všechny tři složky barev současně, nicméně je potřeba brát v potaz trojnásobné zvětšení matice. Výsledky vyhledávání jsou komplexnější. V případě použití `HSV` barevného systému vyhledávací algoritmus LSI lépe reagoval na drastické změny světelných podmínek nebo naopak na velmi malé světelné změny obrazu. Testování nicméně ukázalo, že toto chování barevného kódování [14] nepřineslo celkové zlepšení vyhledávání neboť cílem bylo vyhledávat vizuálně podobné objekty, pokud možno bez ohledu na světelné změny.

Numerické testování simulačního algoritmu ukázalo zajímavé vlastnosti. Dalo by se např. očekávat, že simulační metoda Monte-Carlo (MC) bude nejlépe fungovat u hranice $75-90$ procent vybraných řádků z původní matice. Nicméně výsledky numerických experimentů na reálné obrazové matici ukázaly, že nejlepší výsledky vyhledávání je dosaženo u hranice $51-60$ procent vybraných řádků matice. Tedy "statisticky stručnější"popis obrazu vedl k lepším výsledkům vyhledávání.

Numerické experimenty dále potvrdily, že konvergence simulačního procesu nezávisí na dimenzi analyzované matice: nezávisí tolik, jestli jsou zkoumaných dokumentů desítky nebo stovky.

Další zajímavou zkušenost přineslo snížení rozlišení testovaných obrázků. Rozlišení obrázků bylo algoritmicky sníženo na hranici $160 * 120$ pixelů. Numerické experimenty ukázaly, že v tomto rozlišení simulační metoda MC pracovala nejen stejně, ale i o něco lépe než s dvojnásobně vysokým rozlišením.

Numerické experimenty dále ukázaly, že použití simulační metody MC bylo úspěšné i po stránce paměťové náročnosti: došlo ke snížení paměťové náročnosti a zároveň se zkvalitnily výsledky vyhledávání. Výrazně se zlepšil (vyhladil, linearizoval) průběh singulárních čísel matice S, což vedlo při experimentech ke snížení hodnoty koeficientu k, viz např. obrázek 5. Tím se snížil i celkový výpočetní čas SVD-LSI části.

Na druhé straně, paralelní výpočet simulační metody MC příkazem `Parfor` trval průměrně přibližně $0.4 + 0.45$ sekundy, ale v některých dokonce případech jen polovinu této hodnoty. Tudíž časová úspora nebyla konstantní. Lze předpokládat, že úspora by byla významnější při analýze rozsáhlejších dat, neboť teoreticky rychlost konvergence simulačního procesu vůbec nezávisí na dimenze analyzované matice (viz [6]).

Numerické experimenty dále ukázaly, že simulační metoda MC lze s výhodou využít i pro automatickou detekci podobných obrázků z videosekvencí. Jednotlivé snímky z videosekvence se uloží např. po 0.5 sekundě do vektoru. Následně z jednotlivých kódovaných snímků sestaví vstupní matice, např. A0, na kterou se aplikuje modifikovaná simulační metoda MC. Navrhovaná modifikace spočívá ve výpočtu podobnosti bez uvažování podílu (váhy) jednotlivých řádků. Tímto postupem získáme s velkou pravděpodobností reference na obrázky, které byly náhodně vybrány. Naopak se s velkou pravděpodobností odfiltrují obrázky , u nichž nedošlo k náhlé, velké změně.

Lukáš Pištělák

7 Reference

[1] Dubi, A., *Monte Carlo Applications in Systems Engineering*, Ben Gurion University of the Negev, Beer-Sheva,Israel.

[2] Dostál, Zdeňek, *Lineární algebra*, Vysoká škola báňská – Technická univerzita Ostrava, 2011.

[3] Kotas, Petr, *Efektivni implementace singulárniho rozkladu matice*, Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2007.

[4] Krátký, Michal, *Využití SVD pro indexování latentní sémantiky*, Department of Computer Science, VŠB-Technical University of Ostrava, Czech Republic.

[5] Praks, Pavel, Machala, Libor, Snášel Václav, *On SVD-Free Latent Semantic Indexing for Iris Recognition of Large Databases*, Department of Computer Science, VŠB-Technical University of Ostrava, Czech Republic.

[6] Petros Drineas, Eleni Drinea, Patrick S. Huggins: *An Experimental Evaluation of a Monte-Carlo Algorithm for Singular Value Decomposition*, USA.

[7] Berry WM, Drma c Z, Jessup JR.: Matrices, Vector Spaces, and Information Retrieval. *SIAM Review*;41(2):336-362.,1999.

[8] Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. *SIAM*, 1992.

[9] Muller N, Magaia L, Herbst BM.: Singular Value Decomposition, Eigenfaces, and 3D Reconstructions. *SIAM REVIEW* ;46(3):518545.,2004.

[10] Wikramaratna, Roy: Pseudo-random Number Generation for Parallel Monte Carlo. *A Splitting Approach. from SIAM News*, Volume 33, Number 9.

[11] Sobol,I.M.: *Distribution of points in a cube and approximate evaluation of integrals* U.S.S.R: Zh. Vych. Mat. Mat. Fiz. 7: 784–802.

[12] Furnas, G., et al.:The Vocabulary Problem in Human-System Communication *Communications of the ACM*, 30(11), pp. 964-971, 1987.

[13] Lew, Michael et al.: Content-based Multimedia Information Retrieval: *State of the Art and Challenges ACM Transactions on Multimedia Computing, Communications, and Applications*, pp. 1–19, 2006.

[14] Photoshop's documentation explains that, e.g., "Luminosity: Creates a result color with the hue and saturation of the base color and the luminance of the blend color."Dostupné z: *http://helpx.adobe.com/photoshop.html?content=WSfd1234e1c4b69f30ea53e401031ab64-77e9.html*

[15] Zukas, Anthony, Price, Robert J.:Document Categorization Using Latent Semantic Indexing. *White Paper, Content Analyst Company, LLC*

[16] Dostupné z: *http://eigen.tuxfamily.org/index.php?title=Main*$_{Page}$

[17] Visual Search Technology - App Monetization - Search By Image. *Superfish* , 2011-12-05. Retrieved 2012-10-18.

[18] Dumais, S., Platt J., Heckerman D., and Sahami M.,Inductive Learning Algorithms and Representations For Text Categorization. *Proceedings of ACM-CIKM'98*. 1998.

[19] Deerwester, S., et al, Improving Information Retrieval with Latent Semantic Indexing. *Proceedings of the 51st Annual Meeting of the American Society for Information Science 25*, pp. 36–40, 1988.

A Další experimetny a měření

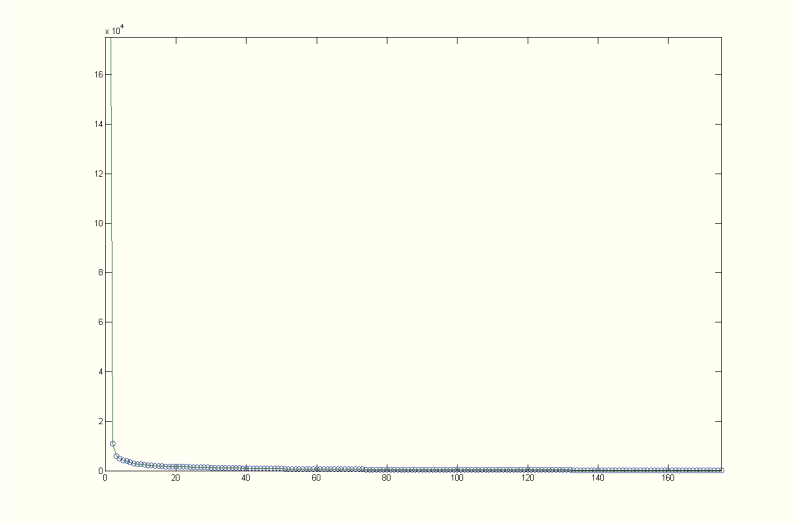
Modifikované maticové násobení $A^T A$, pro jazyk C++ , použito ve spolupráci s výkonnou knihovnou Eigen [16]. V matici Va jsou uloženy sloupce hodnot. Algoritmus jde jednoduše modifikovat na matici typu sparse, řídkou matici, po setřízení hodnot podle indexů, jednoduše přidáme binární vyhledávání. S podmínkou.

```
void MatrixMultiply(int pocetobr,int size) {

        // multiply of matrix; Modificated by 2012. for dense matrix.
        //
        // wrost case;
   VectorXd Bcolj[size];
   int Bcoljindex[size];


   VectorXd Arowi[size];
      int Arowiindex[size];

        // PhotoRecord loaddata = null;
        for (int j = 0; j < pocetobr; j++) {

              Bcolj = va.column(j);
              //Bcolj[m] = Cv[m][j];


              //Bcoljindex[m] = ind.column(j);
              //Bcoljindex[m] = Ci[m][j];



            // }
            for (int i = 0 + j; i < pocetobr; i++) {// pro kazdy
dalsi   prvek jen co je pod diagoanle a vcetne!!


                 Arowi = va.column(i);



                 //Arowiindex = ind.column(i);




              // float [][] Arowi = service.accessMatrix(i);
              double s = 0;

              for (int k = 0; k < size; k++) {
//size=lenght of column;
                 // int hledCislo = Bcoljindex[k];
```

```
                    s += Bcolj[k] * Arowi[r];


                }
                C(i,j) = s;

                // saveSomeelemnts(int C[i][j]) ;// melo by to
mit i v
                // nejhorsich pripadecj 26 MB takze to asi
bude v ramce
                // tezko rict
            }

        }

        // loaddata2.setData(null);
        // loaddata2.setIndexes(null);

        // loaddata.setData(null);
        // loaddata.setIndexes(null);

        for (int i = 0; i < pocetobr - 1; i++) {
            for (int m = pocetobr - 1; m > i; m--) {
                C(i,m) =C(m,i);
            }

        }




    }
end
```

Výpis 12: Maticové násobení pro hustou matici A

Obrázek 18: Query obrázek auto, při k=15 a s=46 procent, zde vidíme ne dobrý výsledek způsobený velkou ztrátou řádků výchozí matice, proto je zobrazeno pozadí s přeskoky.

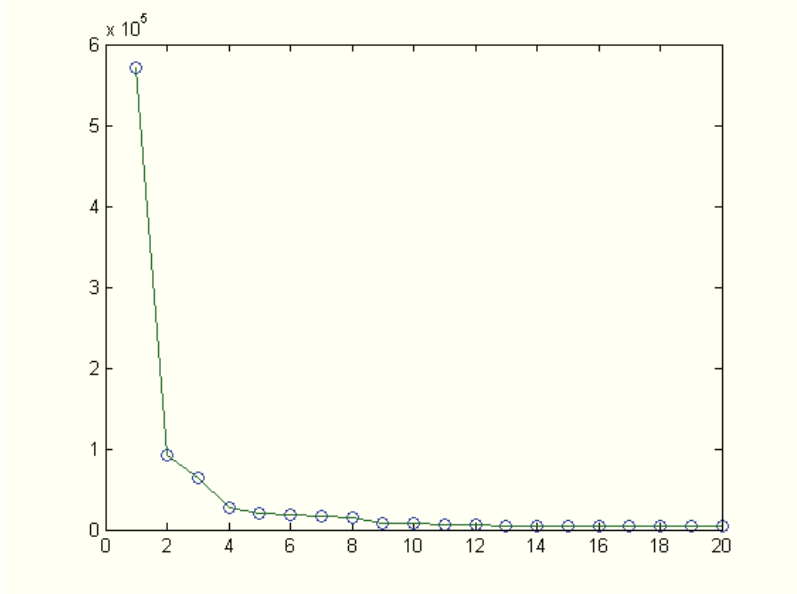


Obrázek 19: Ukázka grafu singulárních čísel, až do hodnoty 160

Obrázek 20: Query člověk, zde je vidět důsledek příliš velkého k=101, s je nastaveno optimálně.

Obrázek 21: Query auto, zde je vidět důsledek příliš velkého k=50, s je nastaveno optimálně, nicméně je zde vidět přeskakování mezi auty a pozadím.

Obrázek 22: Query traktor, skvělý výsledek, s je nastaveno optimálně, k=20



Obrázek 23: Graf singulárních čísel, query traktor, skvělý výsledek, s je nastaveno optimálně, k=20